

**课程设计报告**

**题目：基于SAT的数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： 计科2011班**

**学 号： U202015084**

**姓 名： 张文浩**

**指导教师： 许贵平**

**报告日期： 2021.9.7**

**计算机科学与技术学院**

**任务书**

**设计内容**

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于DPLL算法实现一个完备SAT求解器，对输入的CNF范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

**设计要求**

要求具有如下功能：

**(1)输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。

**(2)公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。

**(3)DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。

**(4)时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。

**(5)程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。

**(6)SAT应用：**将数独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。

**参考文献**

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

[2]Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

[4]Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243

[5]360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

[7] Ins Lynce and Jol Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale. Springer, 2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

1. Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

**目录**

**任务书** I

**1引言** 1

1.1课题背景与意义 1

1.1.1 课题背景 1

1.1.2 研究意义 1

1.2 国内外研究现状 2

1.3 课程设计的主要研究工作 3

**2 系统需求分析与总体设计** 4

2.1系统需求分析 4

2.2系统总体设计 4

**3 系统详细设计** 6

3.1 有关数据结构的定义 6

3.1.1 有关数据结构的具体定义 6

3.1.2 各数据结构之间的关联 7

3.2 主要算法设计 7

**4 系统实现与测试** 13

4.1 系统实现 13

4.1.1 编程环境 13

4.1.2 用户需求 13

4.1.3 数据结构定义 14

4.1.4 模块实现 14

4.2 系统测试 22

**5 总结与展望** 29

5.1 全文总结 29

5.2 工作展望 29

**6 体会** 30

**参考文献** 31

**附录**  32

**1引言**

**1.1课题背景与意义**

* + 1. **课题背景**

布尔表达式是由布尔变量和运算符（NOT , AND ,OR）所构成的表达式。

如果对于变量的某个true,false赋值，使得一个布尔表达式的值为true，则该布尔表达式是可满足的。例如布尔公式 A = ((NOT x) AND y) OR ( x AND (NOT z))，当 x = false, y = true, z = false时，该布尔表达式值为true，则表达式A就是可满足的。可满足性问题就是判定一个给定的合取范式的布尔公式是否是可满足的。

已知的NP-complete问题多达几百个，但作为这些问题的“祖先”，历史上第一个被证明的NP-complete问题是来自于布尔逻辑的可满足性问题（SATISFIABLITY problem），简称为SAT。

SAT的问题被证明是NP难解的问题。目前解决该问题的方法主要有完备的方法和不完备的方法两大类。完备的方法优点是保证能正确地判断SAT问题的可满足性，但其计算效率很低，平均的计算时间为多项式阶，最差的情况计算时间为指数阶，不适用于求解大规模的SAT问题。不完备的方法的优点是求解的时间比完备的方法快得多，但在很少数的情况下不能正确地判断SAT问题的可满足性。传统的方法有：枚举法、局部搜索法和贪婪算法等，但由于搜索空间大，问题一般难以求解。对于像SAT一类的NP难问题，采用一些现代启发式方法如演化算法往往较为有效。

* + 1. **研究意义**

SAT问题是逻辑学的一个基本问题，也是当今计算机科学和人工智能研究的核心问题。工程技术、军事、工商管理、交通运输及自然科学研究中的许多重要问题，如程控电话的自动交换、大型数据库的维护、大规模集成电路的自动布线、软件自动开发、机器人动作规划等，都可转化成SAT问题。因此致力于寻找求解SAT问题的快速而有效的算法，不仅在理论研究上而且在许多应用领域都具有极其重要的意义。

可满足性问题是计算机科学领域和人工智能等领域中的重要研究对象。但是 其求解算法的时间开销和空间开销却异常惊人。对于一个含有n个命题逻辑变量的合取范式来说，如果使用穷举法来罗列所有真值指派进行求解，虽然在理论上是可行的，但算法的时间复杂度却是为O(2n)级的，计算机如果采用这种方式进行求解将负担不起如此大的开销。

SAT问题为NP完全问题，NP完全问题排在七大数学难题之首，在计算复杂性理论中具有非常重要的地位，一方面因为它有着极大的理论价值并且非常难解，另一方面是一旦被破解以后，在诸多的工程领域里还可以得到广泛的应用。由于SAT问题是NP完全问题，它如果能够得到高效解决，那么一定可以高效地解决所有其它NP完全问题，这是因为所有的NP完全问题都能在多项式时间内进行相互转化，即所有的NP完全问题都能够在多项式时间内转换为可满足性问题。

研究能够解决SAT问题的高效算法在理论上有着重要的意义。SAT问题的应用领域非常广泛，例如在数学研究和应用领域，它能用来解决旅行商（Traveling Salesman Problem，TSP）和逻辑算术问题；在计算机和人工智能（Artificial Intelligence）领域中，它能解决CSP（约束满足问题）问题、语义信息的处理和逻辑编程等问题；在计算机辅助设计领域中，它能很好的解决任务规划与设计、三维物体识别等问题。许多的实际问题如人工智能、积木世界规划问题、数据库检索、Job shop排工问题、超大规模集成电路设计和图着色都可转换为 SAT 问题进行求解。不仅如此，一些实际问题通常还具有一定的结构特性，可以据此来对具有某一类结构特性的实际应用问题进行求解，进而得到解决该类问题的一种通用解法。

**1.2 国内外研究现状**

DPLL算法是由Davis和Putnam 等人在1960年提出，其它的完备算法大都是在DPLL算法的基础上衍生出来的，是对DPLL算法的改进。S.A.Cook在1971年证明了SAT问题是NP完全问题，这削弱了许多学者研究SAT问题的兴趣，从而导致了SAT问题在很长的一段时间里都没有得到较好的重视，发展非常缓慢，研究成果较少。但是1996年以后，很多国家都相继举办了一些SAT竞赛和研讨会，这使得越来越多的人开始关注并研究SAT问题。SAT协会是目前推动SAT问题理论和应用进展的主要驱动力量，其Satlive网站随时更新SAT研究动态，发布了一系列有关会议、竞赛、技术报告、论文、图书等信息；每年举办一次SAT理论和应用国际学术会议，目前已召开16届；SAT国际竞赛始于2002年，每隔两年或一年举办，2016年成功举办了第10届，汇聚了大批优秀的SAT求解器，影响力很大。

冲突驱动子句学习（Conflict Driven Clause Learning，简称CDCL）算法主要框架基于DPLL，是目前最重要的SAT求解算法，在冲突分析与子句学习、非时许回溯、重启、数据结构等方面做了一系列改进。当前代表性的CDCL求解器包括GRASP、Chaff、MiniSAT、Lingeling、Glucose、Sparrow等。其中Moskewicz等人提出的Chaff算法集合了laziness、线性学习和一系列启发式策略，强调“快速和低廉”的低负荷决策方法，是SAT问题求解算法的重要突破，形成流行求解器的相关思想。Lingeling、Glucose、Sparrow等近年来在SAT国际竞赛中屡获佳绩。国内也涌现出了许多高效的求解算法，如1998年作者梁东敏提出了改进的子句加权WSAT算法，2000年金人超和黄文奇提出的并行Solar算法，2002 年作者张德富在文献中，提出模拟退火算法。

尽管SAT算法已经取得了举足轻重的改进，但是仍有一些问题没有得到高效的解决，已经解决的问题可能还存在更好的求解算法，因此研究并实现高效率的求解算法仍是当前要解决的中心问题之一。

**1.3 课程设计的主要研究工作**

（1）了解SAT问题的背景、研究意义以及国内外研究现状。

（2）掌握SAT问题相关的基本理论与知识

（3）设计数据结构，读取cnf文件，基于DPLL算法实现一个SAT求解器，能够求解一定规模的SAT问题。

（4）改进数据结构和算法等，提高时间效率

（5）将数独问题转化为SAT问题，并利用SAT求解器生成以及求解数独，整合为一个数独交互游戏。

**2 系统需求分析与总体设计**

**2.1 系统需求分析**

**（1）输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。

**（2）公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；遍历内部结构逐行输出与显示每个子句。

**（3）DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。

**（4）时间性能的测量：**基于相应的时间处理函数，记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。

**（5）程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现。

**（6）SAT应用：**将数独游戏问题转化为SAT问题，并集成到上面的求解器进行问题求解，游戏可玩，具有一定的/简单的交互性。

**2.2 系统总体设计**

该系统包括主控、交互与显示模块；SAT模块；SUDOKU模块。

**主控、交互与显示模块：**显示交互环境，读取用户输入，显示执行结果。

**SAT模块：**

1. 基于特定数据结构，读取并解析cnf文件。
2. 遍历内部结构逐行输出与显示每个子句。
3. 调用DPLL算法对算例进行求解，并将答案保存为res文件。
4. 将数据结构中的内容保存为cnf文件。‘

**数独模块：**

1. 生成数独终盘，将数独转化为SAT算例，并利用挖洞法生成待求解数独。
2. 在生成的待求解数独中填入或删除数字，或者查看答案。

系统的结构图如下：

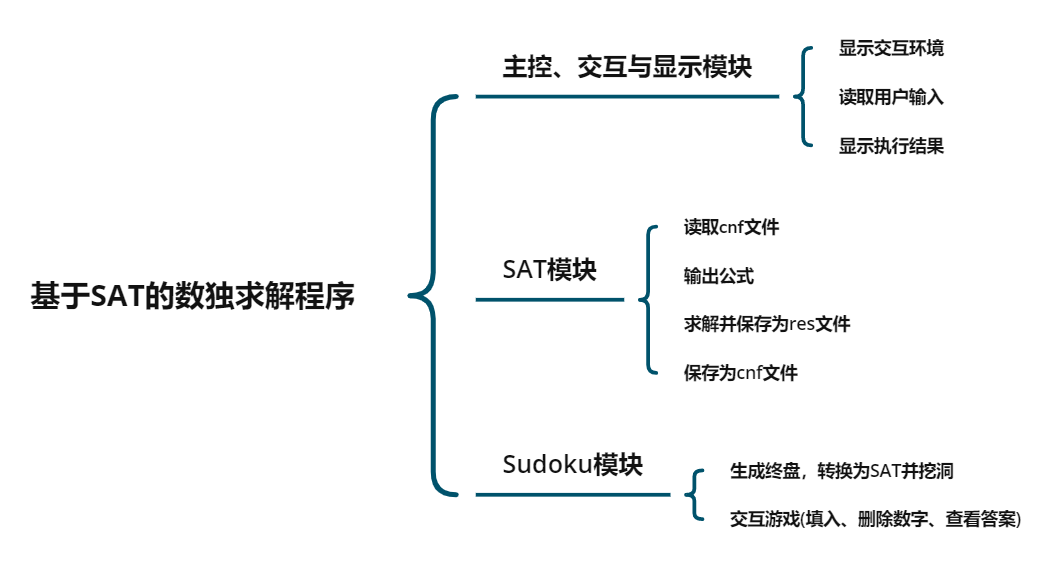


图2-1 系统总体设计结构图

**3系统详细设计**

**3.1 有关数据结构的定义**

系统在整体上采用邻接表结构，需要储存的主要有文字类型、子句类型、子句集(公式)类型。

**3.1.1 有关数据结构具体定义**

所用数据类型以及所储存数据项：

①varnode类型（Varnode为对应指针）

表3-1 varnode类型所储存数据项

|  |  |  |
| --- | --- | --- |
| 数据项类型 | 数据项名称 | 数据项作用 |
| int | data | 储存文字 |
| struct varnode\* | next | 指向本子句下一个文字 |

②clause类型（Clause为对应指针）

表3-2 clause类型所储存数据项

|  |  |  |
| --- | --- | --- |
| 数据项类型 | 数据项名称 | 数据项作用 |
| int | count\_of\_var | 储存当前本子句中文字数量 |
| struct varnode\* | head | 指向本子句第一个文字 |
| struct clause\* | next | 指向下一个子句 |

③total类型（Total为对应指针）

表3-3 total类型所储存数据项

|  |  |  |
| --- | --- | --- |
| 数据项类型 | 数据项名称 | 数据项作用 |
| int | varcount | 储存公式中文字数量 |
| int | clacount | 储存公式中子句数量 |
| int | singleclause | 储存当前公式中单子句个数 |
| int | emptyclause | 储存当前公式中空子句个数 |
| int | count\_of\_clause | 储存当前公式中子句个数 |
| struct clause\* | clahead | 指向第一个子句 |

**3.1.2 各数据结构之间的关联**

各数据结构之间的关联如下图所示：

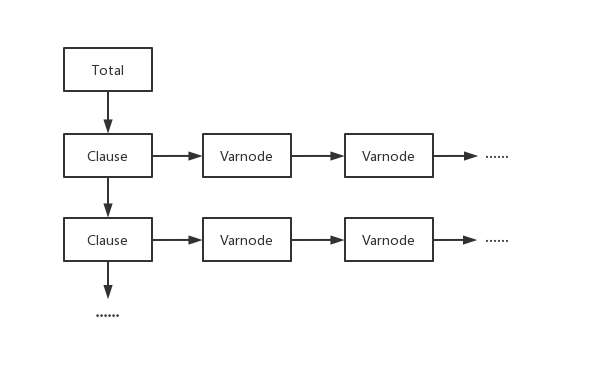
****

图3-1 各数据结构之间的关联图

**3.2 主要算法设计**

(1)交互模块未涉及算法

(2)SAT模块：主要为DPLL算法。

DPLL算法是基于树/二叉树的回溯搜索算法，主要使用两种基本处理策略：

单子句规则。如果子句集*S*中有一个单子句*L*,那么*L*一定取真值，于是可以从*S*中删除所有包含*L*的子句（包括单子句本身），得到子句集*S*1，如果它是空集，则*S*可满足。否则对*S*1中的每个子句，如果它包含文字*¬L*,则从该子句中去掉这个文字，这样可得到子句集合*S*2。*S*可满足当且仅当*S*2可满足。单子句传播策略就是反复利用单子句规则化简*S*的过程。

分裂策略。按某种策略选取一个文字*L*.如果*L*取真值，则根据单子句传播策略，可将*S*化成*S*2；若*L*取假值（即*¬L*成立）时，*S*可化成*S*1.

本次课程设计对变元选取策略进行了优化，优化策略为统计每个变元的所有所在子句的权值之和，其中每个子句的权值为1/2的子句变元数次方，由此可见，变元数越少，最终权值越大，我们选取权值最大的变元，说明该变元出现次数比较多而且位于变元数较少的子句的概率较大，选取该变元可以减少回溯次数。

该程序中的DPLL算法的流程图如下图所示：

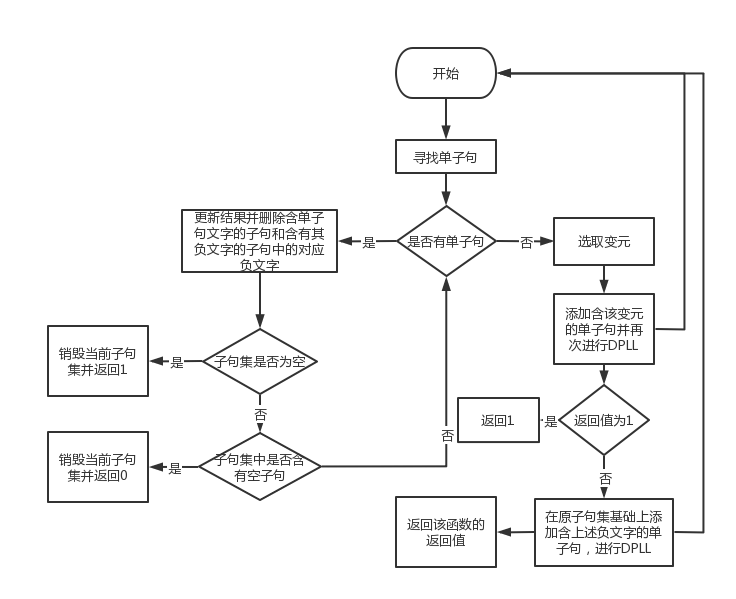


图3-2 DPLL算法流程图

(3)SUDOKU模块

①生成数独终盘算法

算法主要思想为依次在各行依照数独顺序填入从1-9的每个数，先填完各行的1，再填各行的2，以此类推。在填写某行某数时，先生成一个随机的列序号数组，然后遍历该数组，如果在某一列可以填入该数字，就填入该数字，进行该数字下一行的填写和从第一行开始下一个数字的填写，之后回溯，复原该数字。

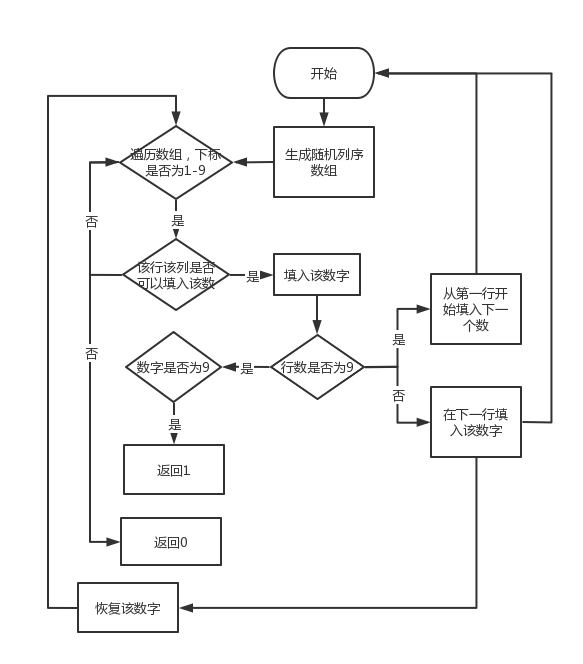
流程图如下：

图3-3 生成数独终盘算法流程图

②数独转化为SAT算法

变元可按语义编码为1～9之间数字构成的三位整数*ijk*，*i*, *j*, *k*∈{1,2,…,9}，其中*i*表示单元格的行号，*j*表示单元格的列号，*k*表示单元格<*i*, *j*>填入的数字为*k*。如163变元表示第1行6列填入3；负文字 -452表示第4行5列不填入2。这样编码共有729个变元。

数独游戏的基本要求是：每个单元格只能填入唯一一个数字。以单元格<1,1>例，这可以表示为如下子句：

111 112 113 114 115 116 117 118 119 0

-111 -112 0

-111 -113 0

……

-118 -119 0

上述表示中，每个子句的末尾的0表示结束标记；第一个子句的含义是单元格<1,1>可填入至少一个数字；后面的子句集共同表示只能填入一个数字，子句-111 -112 0表示不能同时填1与2；其它类推。按这种方式需要对81个单元格进行类似表示，得到对应的子句集。

行约束要求每行需要填入1～9中的每个数字，且每个数字只出现一次。以第1行为例可表示为（此处在每个子句后加入注释，说明子句的含义）：

111 121 131 141 151 161 171 181 191 0 第1行含有1

112 122 132 142 152 162 172 182 192 0 第1行含有2

… …

119 129 139 149 159 169 179 189 199 0 第1行含有9

-111 -121 0 前两格不同时为1

-111 -131 0 第1与第3格不同时为1

… …

-111 -191 0 第1与第9格不同时为1

… …

列约束仿照行约束易于表示为对应子句集，同学们可自行写出。

对于3×3的盒子约束，以左上角的盒子为例进行说明，其子句集可表示如下：

111 121 131 211 221 231 311 321 331 0 包含1

112 122 132 212 222 232 312 322 332 0  包含2

… …

119 129 139 219 229 239 319 329 339 0 包含9

-111 -211 0 11格与21格不同时为1

-111 -311 0 11格与31格不同时为1

-111 -121 0 11格与12格不同时为1

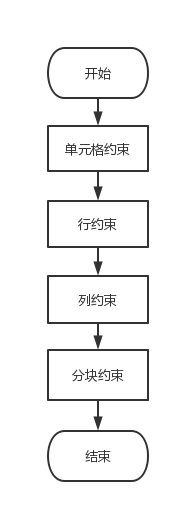
… …

最后，对于每个具体的数独游戏，已经填入了部分提示数，如图2.3中的左图，每个提示数可表示为一个单子句，如第2行3列填入5，对应单子句如下：

235 0

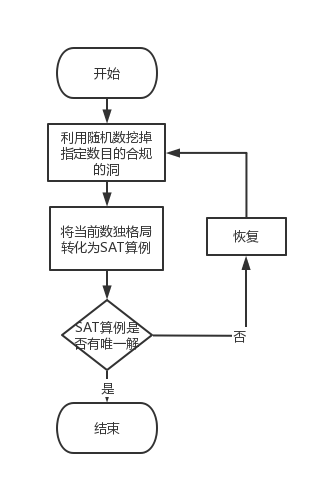
SAT公式CNF文件中，一般变元是从1进行连续编码的，可以将上述语义编码转换为自然顺序编码，公式为：*ijn* → (*i*-1)\*81+(*j*-1)\*9+*n*；当按自然编码对数独游戏对应的CNF公式求解后，可设计逆变换公式将解解析为对应的游戏填充方案，完成填充。

根据上面的分析，数独约束生成CNF子句集易于用多重循环结构实现。

图3-4 数独转化为SAT算法流程图

③挖洞法生成待求解数独算法

算法主要思想为，利用随机数挖掉指定数目的符合数独规则的洞，将目前的数独格局转换为SAT算例并求解，如果有唯一解说明符合要求，如果没有唯一解，说明违反了数独规定，需先恢复数独，再重新挖洞，重复上述流程。

图3-5 挖洞法生成待求解数独算法流程图

**4 系统实现与测试**

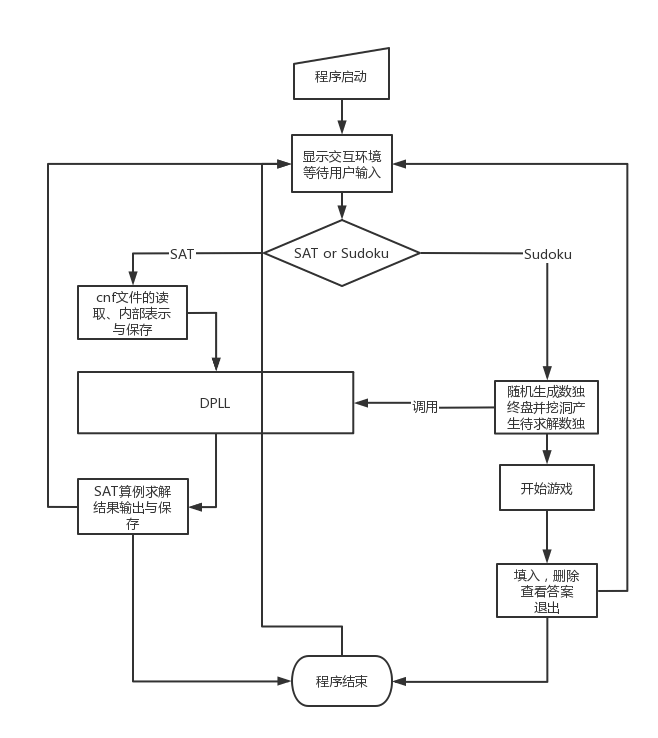
## 4.1系统实现

**4.1.1 编程环境**

Windows环境下使用Microsoft Visual Studio 2019

**4.1.2 用户需求**

该系统主要实现cnf文件的读入与保存，cnf文件的输出，SAT算例的求解；数独的生成、求解与交互。系统主控流程图如下：

图4-1 系统主控流程图

**4.1.3 数据结构定义**

typedef struct varnode {

int data; //数据

struct varnode\* next; //下一个文字

}varnode, \* Varnode; //文字类型

typedef struct clause {

int count\_of\_var; //文字数量

struct varnode\* head; //第一个文字

struct clause\* next; //第一个子句

}clause, \* Clause; //子句类型

typedef struct total

{

int varcount; //文字数

int clacount; //公式数

int singleclause; //单子句数量

int emptyclause; //空子句数量

int count\_of\_clause; //目前子句数量

struct clause\* clahead; //第一个子句

}total, \* Total; //公式类型

**4.1.4 模块实现**

**交互模块：**

该模块首先打印主界面，显示功能菜单，接受用户输入的选项。然后将输入转化合适的参数，调用适当的模块提供功能，并负责输出提示信息。

**SAT模块：**

①cnf保存解析部分：

1. Total fileload(Total SATList, char\* filename);

该函数以子句集和文件名作为参数，读取指定cnf文件中的内容，建立公式的内部表示，并将其保存在子句集中，返回创建的子句集。

时间复杂度为O（mn）（m为子句数，n为每个子句中的变元数）

空间复杂度为O（mn）

1. status Traverse(Total SATList);

该函数以子句集作为参数，遍历已经创建好的子句集中的内容，将其以命题公式的形式打印出来，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（1）

②DPLL部分：

1. int findsingleclause(Total SATList);

该函数以子句集作为参数，遍历子句集，寻找单子句，并将单子句中的文字返回；如果没有找到单子句(子句集成员变量singleclause为0)，返回0。

时间复杂度为O（m）

空间复杂度为O（1）

1. status deleteclause(Total SATList, int temp);

该函数以子句集和目标文字作为参数，遍历子句集，寻找含有目标文字的子句并将其从子句集中删除，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（1）

1. status deleteword(Total SATList, int temp);

该函数以子句集和目标文字作为参数，遍历子句集，在每个子句中寻找目标文字的负文字，如果有，就将其删除，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（1）

1. int judgeemptytotal(Total SATList);

该函数以子句集作为参数，判断当前子句集中的成员变量count\_of\_clause是否为0，如果为0，则返回1，否则返回0。

时间复杂度为O（1）

空间复杂度为O（1）

1. int judgeemptyclause(Total SATList);

该函数以子句集作为参数，判断当前子句集的成员变量emptyclause是否大于0，如果其大于0，则返回1，否则返回0。

时间复杂度为O（1）

空间复杂度为O（1）

1. int getword1(Total SATList);

该函数为获取决策变量的第一个算法，以子句集作为参数，统计所有正负文字的出现次数，返回出现次数最多的文字。

时间复杂度为O（mn）

空间复杂度为O（v）（v为变元数量）

1. int getword2(Total SATList);

该函数为获取决策变量的第二个算法，以子句集作为参数，返回第一个子句中的第一个文字。

时间复杂度为O（1）

空间复杂度为O（1）

补. int getword3(Total SATList);

该函数为获取决策变量的第二个优化算法，对于所有正负变元，将其所在的子句的变元数作为1/2的幂次，然后对其所在的所有子句都进行相同操作并求和，得到每个变元对应的权重，选出其中的最大值对应的变量并返回。

1. status addclause(Total SATList, int temp);

该函数以子句集和决策变量作为参数，将以决策变量作为唯一文字的单子句添加到传入的子句集的末尾，用于下一个DPLL，返回值为函数执行状态。

时间复杂度为O（1）

空间复杂度为O（1）

1. Total Duplicate(Total SATList);

该函数以子句集作为参数，产生当前子句集的副本，用于回溯，返回生成的子句集副本。

时间复杂度为O（mn）

空间复杂度为O（mn）

1. Total destroy(Total SATList);

该函数以子句集作为参数，依次销毁子句集中的每个子句中的每个文字以及每个子句，最后将子句集本身销毁并置空，返回销毁后的子句集。

时间复杂度为O（mn）

空间复杂度为O（1）

1. status DPLL1(Total SATList);

status DPLL2(Total SATList);

分别使用了不同的决策变量获取算法的两个DPLL算法，该函数为核心函数，首先进行单子句传播策略，如果子句集为空就返回1，如果含有空子句就返回0。然后复制当前子句集，获取决策变量并添加，再次进行单子句策略，由这次的返回值决定是否使用副本添加负决策变量并再次进行单子句策略。

时间复杂度为O（mnlogv）

空间复杂度为O（mnlogv）

③文件保存部分：

1. status saveanswer(Total SATList,int dresult, double time);

该函数以子句集，DPLL执行状态变量，DPLL执行时间作为参数，将DPLL执行状态（是否有解），结果数组的内容，函数执行时间写入同名res文件，返回值为函数执行状态。

时间复杂度为O（v）

空间复杂度为O（1）

1. status savecnf(Total SATList);

该函数以子句集作为参数，将子句集中的内容写入cnf文件，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（1）

**Sudoku模块：**

①转换数独部分：

1. status convertresult(int sudoku[][10]);

该函数以数独数组作为参数，遍历结果数组，将结果按转换关系转换成对应

的行列数据并填入数独中，返回值为函数执行状态。

时间复杂度为O（1）

空间复杂度为O（1）

1. status printsudoku(int sudoku[][10])；

该函数以数独数组作为参数，将数独数组按照数独的形式输出，返回值为函数执行状态。

时间复杂度为O（1）

空间复杂度为O（1）

②生成数独部分：

1. status judgenum(int sudoku[][10], int row, int col, int n);

该函数以数独数组，行号，列号，待填入数据作为参数，判断当前位置是否可以填入该数据，即判断该行、该列以及所在的九宫格中是否已经含有该数据，如果可以填入，返回1，否则返回0。

时间复杂度为O（1）

空间复杂度为O（1）

1. Total initial(Total SATList);

该函数以子句集作为参数，完成子句集成员变量的初始化，为数独数据转换成SAT算例作准备，返回所操作的子句集。

时间复杂度为O（1）

空间复杂度为O（1）

1. status createnode(Total SATList,int temp);

该函数以子句集和目标文字作为参数，在当前子句集的最后一个文字之后创建一个含有目标文字的新文字结点，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（1）

1. status createorigin(Total SATList, int sudoku[][10]);

该函数以子句集和数独数组作为参数，对数独数组中的每一个不为0的元素，按约束规则和转换关系，调用createnode函数创建结点，返回值为函数执行状态。

时间复杂度为O（v）

空间复杂度为O（v）

1. status createsingle(Total SATList, int sudoku[][10]);

该函数以子句集和数独数组作为参数，对数独数组中的每一个单元格，按约束规则和转换关系，调用createnode函数创建结点，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（v）

1. status createrow(Total SATList, int sudoku[][10]);

该函数以子句集和数独数组作为参数，对数独数组中的每一行，按约束规则和转换关系，调用createnode函数创建结点，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（v）

1. status createcol(Total SATList, int sudoku[][10]);

该函数以子句集和数独数组作为参数，对数独数组中的每一列，按约束规则和转换关系，调用createnode函数创建结点，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（v）

1. status createblock(Total SATList, int sudoku[][10]);

该函数以子句集和数独数组作为参数，对数独数组中的每一宫，按约束规则和转换关系，调用createnode函数创建结点，返回值为函数执行状态。

时间复杂度为O（mn）

空间复杂度为O（v）

1. Total createcnf(Total SATList, int sudoku[][10]);

该函数以子句集和数独数组作为参数，调用上述创建约束函数，将数独转换为SAT算例，返回创建好的子句集。

时间复杂度为O（mn）

空间复杂度为O（v）

1. status createsudoku(Total SATList, int sudoku[][10]);

该函数为第一个生成数独终盘函数，以子句集和数独数组作为参数，首先填入11个符合数独规则的数，然后将目前的数独格局转化为SAT算例并调用DPLL函数求解，再将结果转换为数独，返回值为函数执行状态。

时间复杂度为O（mnlogv）

空间复杂度为O（mnlogv）

1. status createsudoku2(int sudoku[][10],int row,int num);

该函数为第二个生成数独终盘函数，以数独数组，行号，待填入数据作为参数。依次在各行依照数独顺序填入从1-9的每个数，先填完各行的1，再填各行的2，以此类推。在填写某行某数时，先生成一个随机的列序号数组，然后遍历该数组，如果在某一列可以填入该数字，就填入该数字，进行该数字下一行的填写和从第一行开始下一个数字的填写，之后回溯，复原该数字，返回值为函数执行状态。

时间复杂度为O（1）

空间复杂度为O（a）（a为递归次数）

1. void DPLLPRO(Total SATList);

该函数为改进的DPLL算法，在原DPLL算法的基础之上，取消返回值，用一全局变量储存所有解的个数，在单子句策略结束选取决策变量之后，先进行正文字的DPLL，再接着进行负文字的DPLL。如果再单子句策略过程中当前子句集为空就将解的个数加一。如果当前解的个数大于1就直接返回。

时间复杂度为O（mnlogv）

空间复杂度为O（mnlogv）

1. status digholes(int sudoku[][10], int count,int holes[][10]);

该函数为挖洞生成待求解数独的函数，以数独数组，需要挖洞的数量，标记数组作为参数，利用随机数挖掉指定数目的符合数独规则的洞并标记相应位置，将目前的数独格局转换为SAT算例并调用DPLLPRO函数求解，如果有唯一解说明符合要求，如果没有唯一解，说明违反了数独规定，需先恢复数独，再重新挖洞，重复上述流程。

时间复杂度为O（mnlogv）

空间复杂度为O（mnlogv）

**函数调用关系：**

各函数间的调用关系如图4-2所示，图中A指向B代表在A中调用B。

addclause

duplicate

findsingleclause

deleteclause

deleteword

getword

destory

DPLL

DPLLPRO

createsudoku

convertresult

createcnf

initial

createorigin

createsingle

createrow

createcol

createblock

createnode

digholes

printsudoku

judgenum

图4-2 函数调用关系

## 4.2系统测试

主要测试以下内容：

（1）cnf解析：读取cnf文件并建立内部存储。

（2）SAT求解：求解用cnf文件表示的sat问题。

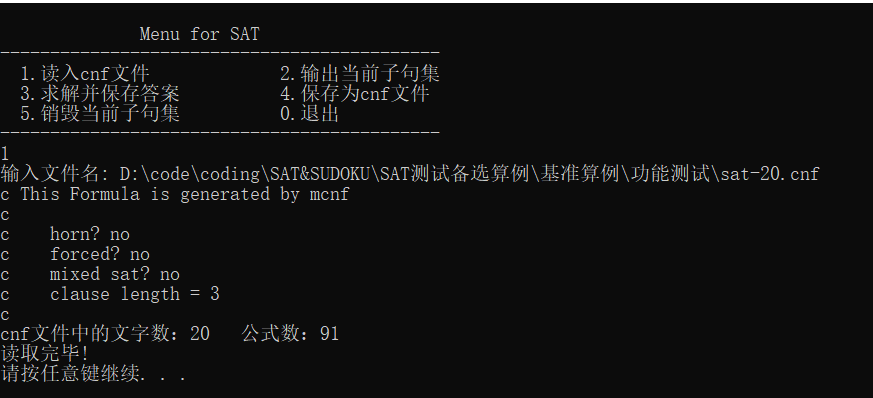
（3）生成数独：随机自动生成可玩数独。

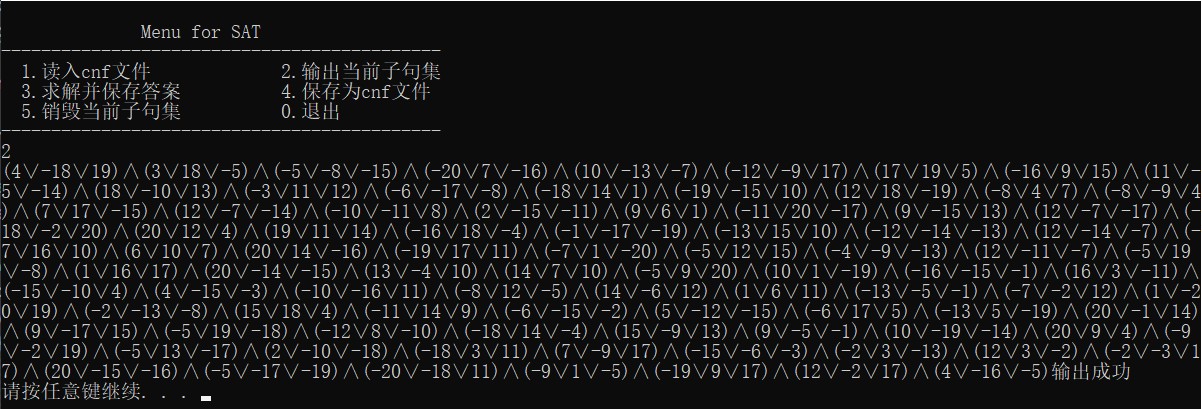
（4）数独交互：填入、删除数字，查看答案

**功能测试：**

1. cnf解析：

测试方法为读取cnf文件并输出文件。读取文件为图4-3，输出文件为图4-4。

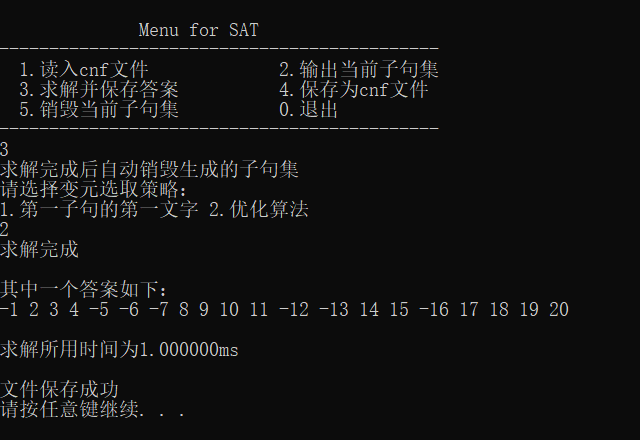
图4-3 读取文件

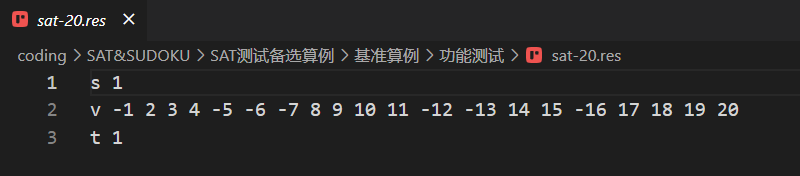
图4-4 输出文件

从图4-4中可以看到正确的输出了cnf文件的公式表示，说明cnf解析部分功能正常，可以正确的建立公式的内部表示。

1. SAT求解：

测试方法为求解建立的子句集、输出解的情况并保存为res文件，如图4-5，图4-6所示。

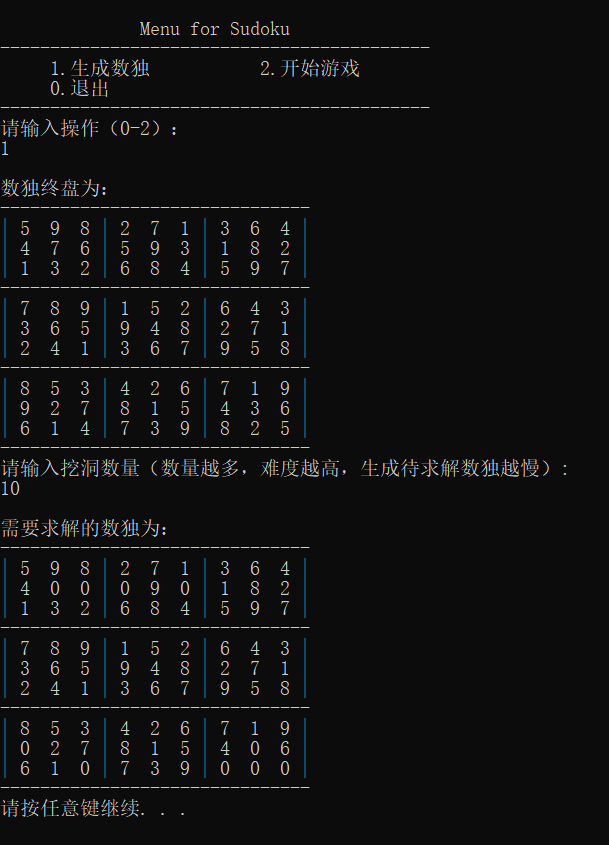
图4-5 求解子句集

图4-6 保存为res文件

从图4-5可以验证，得出的解是正确的，图4-6中可以看出得出的解按照格式输出到了res文件中，说明该部分功能正常。

1. 生成数独

测试方法为随机生成数独终盘并挖掉一定数量的洞，如图4-7所示。

图4-7 随机生成数独终盘并挖洞

从图4-7可以看出生成的数独终盘和挖洞之后的数独均符合数独规则，说明该部分功能正常。

1. 数独交互

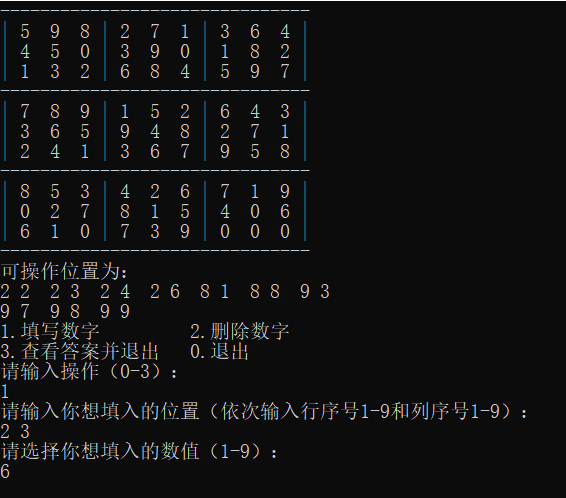
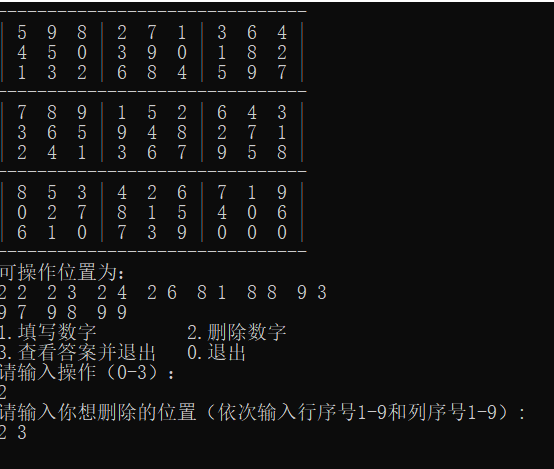
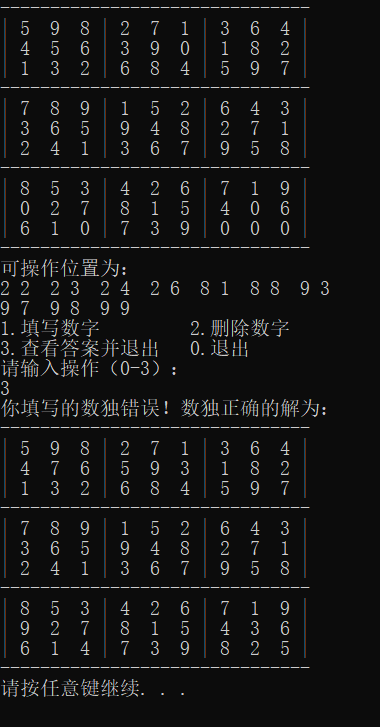
测试方法为对生成的数独进行一系列操作，如图所示。

图4-8 填入数字

图4-9 删除数字

图4-10 查看答案

从图中可以看出数独交互部分功能正常。

（5）测试结果：

对每一部分都进行多次上述测试，测试结果全部正确。可以得知每一部分都能很好地完成预定的任务，功能测试完成。

**性能测试：**

性能测试主要测试SAT模块，为了体现出性能的变化，采用对比的方式进行测试。在求解时可以选取两个不同的变元选取策略。

算例选取：本次测试一共选取19个算例，其中可满足算例有15个，不可满足算例有4个；小型算例有9个，中型算例有7个，大型算例有3个（包括不可满足算例）。

算例测试数据如下：

表4-1 小型算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算例名称 | 变元数量 | 子句数量 | 策略1用时(ms) | 策略2用时(ms) | 优化率 |
| sat-20.cnf | 20 | 91 | 1 | <1 | 100.00% |
| ais10.cnf | 181 | 3151 | 3044 | 35 | 98.85% |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 20 | 1532 | 111 | 19 | 82.88% |
| problem8-50.cnf | 50 | 300 | 4 | 3 | 25.00% |
| problem9-100.cnf | 100 | 200 | >100000 | 19 | 100.00% |
| problem11-100.cnf | 100 | 600 | 12 | 4 | 75.00% |
| tst\_v25\_c100.cnf | 25 | 100 | 2 | 1 | 50.00% |

表4-2 中型算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算例名称 | 变元数量 | 子句数量 | 策略1用时(ms) | 策略2用时(ms) | 优化率 |
| tst\_v200\_c210.cnf | 200 | 320 | 5 | 4 | 25.00% |
| bart17.shuffled-231.cnf | 231 | 1166 | 56 | 28 | 50.00% |
| sud00012.cnf | 232 | 1901 | 33 | 1825 | -5430.30% |
| sud00079.cnf | 301 | 2810 | 121 | 2333 | -1828.10% |
| sud00082.cnf | 224 | 1762 | 13 | 2572 | -19684.62% |
| sud00861.cnf | 297 | 2721 | 10467 | 20569 | -96.51% |

表4-3 大型算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算例名称 | 变元数量 | 子句数量 | 策略1用时(ms) | 策略2用时(ms) | 优化率 |
| eh-dp04s04.shuffled-1075.cnf | 1075 | 3152 | >100000 | 7388 | 100.00% |
| e-par32-3.shuffled-3176.cnf | 3176 | 10297 | >100000 | 47688 | 100.00% |

表4-4 不可满足算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算例名称 | 变元数量 | 子句数量 | 策略1用时(ms) | 策略2用时(ms) | 优化率 |
| u-problem7-50.cnf | 50 | 100 | 159 | 9 | 94.34% |
| unsat-5cnf-30.cnf | 30 | 420 | 266 | 73 | 72.56% |
| u-dp04u03.shuffled-825.cnf | 825 | 2411 | 74267 | 1075 | 98.55% |
| tst\_v10\_c100.cnf | 10 | 100 | 0 | 0 | 0 |

从表中测试结果可以看出，对于小型算例，改进后的算法有较好的优化效果，虽然有部分负优化，但占少数；对于中型算例，改进算法的优化效果很差，有多于一半的算例发生了负优化，经分析cnf原始文件，可能与子句中文字的个数和子句的顺序有关，越是杂乱无章、长短不一的子句集才能获得比较好的优化效果；对于大型算例，由于数据结构、算法和电脑性能的局限，仅仅选取了几个算例进行测试，虽然运行时间较长，但是获得了非常好的优化效果；对于不满足算例，由于求解困难，仅选取了几个算例进行测试，也获得了比较好的优化效果。

测试结果：

无论是可满足算例还是不可满足算例，在整体上都获得了比较好的优化效果，小型算例、大型算例、不可满足算例的优化效果更好，基本达到了性能优化的目的。

# 5.总结与展望

## 5.1全文总结

本次课程设计所完成的的主要任务如下：

（1）明确任务与需求，设计数据结构、程序框架。

（2）分模块编写函数。

（3）调试程序，修改已经出现的错误。

（4）优化数据结构或者算法，实现效率提升。

（5）将所有模块整合成一个交互系统。

## 5.2工作展望

在今后的研究中，围绕以下几个方面开展研究：

（1）改进数据结构。经过本次任务，发现普通的邻接表结构具有很大的弊端，如内存使用过多而无法及时得到释放导致栈溢出、需要复制一份副本来完成回溯，空间开销较大、删除子句和文字比较困难等；需要设计一个更高效的，更符合DPLL算法的较为优秀的数据结构。

（2）优化变元选取算法。本次任务中，发现不同的变元选取策略，会使程序运行时间有着很大的差异。在选取变元时，应该全方位考虑变元数、子句变元的出现频率、子句的长度、子句的排列规则等，对于不同的情况应该对症下药，选取更高效的变元选取策略。

（3）改善交互系统。如实现图形界面，设计更人性化的交互系统等。

1. **体会**

经过本次课程设计，我获得了许多经验和教训。

首先，我第一次体验了自己分析问题，获取需求的过程，从零开始一点点地设计自己的数据结构与程序框架。其次，我体会到了模块化程序设计的过程，本次课程设计的模块较为清晰，可以分模块来实现不同的函数，调试的时候也可以把目光集中在某模块的几个函数中，节约了大量时间，最后将它们全部整合，实现一个完备的交互系统。最后，我体会到了独立思考优化程序的过程，由于一开始编写的程序效率低下，所以需要大幅度优化自己的程序，自己查阅文献，询问老师、同学等，最终获得了一个比较满意的结果。

同时也有一些教训。首先一开始我选取了链表数组来实现子句集，但这种结构在回溯时较难处理，我也没有想到一个比较好的算法来实现回溯，而且这种结构在算例规模较大时很容易出现爆栈的情况，于是又将所有数组用链表来代替，较好地解决了上述问题。其次在生成数独终盘的时候，我原本选取的算法是提前给出11个符合数独规则的数，然后根据目前的数独格局转换层SAT算例并求解，

这样不仅不能保证百分百生成具有唯一解的符合数独规定的数独，而且运行速度非常慢。最后在同学的帮助之下将算法改成了基于DFS 的算法，依次把9个1到9个9填入，这样运行速度优化了许多。最后在挖洞法生成数独时，我原本的算法是每挖一个洞就要判断是否具有唯一解，但这样一旦挖洞数目比较多，就会导致速度非常之慢，之后尝试在挖完所有洞之后再判断是否具有唯一解，效率也提升了不少。

最后感谢老师和同学的帮助，让我比较顺利地完成了本次课程设计。

**参考文献**

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

[2]Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

[4]Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243

[5]360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

[7] Ins Lynce and Jol Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale. Springer, 2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. SAT 2015, 223-237360

**附录**

**SAT\_SUDOKU.h的代码实现：**

#ifndef \_SAT\_H\_

#define \_SAT\_H\_

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#include<time.h>

#define OK 1

#define ERROR 0

typedef int status;

typedef struct varnode {

int data; //数据

struct varnode\* next; //下一个文字

}varnode, \* Varnode; //文字类型

typedef struct clause {

int count\_of\_var; //文字数量

struct varnode\* head; //第一个文字

struct clause\* next; //第一个子句

}clause, \* Clause; //子句类型

typedef struct total

{

int varcount; //文字数

int clacount; //公式数

int singleclause; //单子句数量

int emptyclause; //空子句个数

int count\_of\_clause; //目前子句数量

struct clause\* clahead; //第一个子句

}total, \* Total; //公式类型

int allresult;//所有解的个数

int varcount; //文字数

int \*result; //结果数组

char filename[100]; //需要读入的文件名

char savefile[100]; //需要写入的文件名

//SAT相关函数

Total fileload(Total SATList, char\* filename); //读入cnf文件

Total Duplicate(Total SATList); //复制子句集

status Traverse(Total SATList); //遍历子句集

int findsingleclause(Total SATList); //寻找单子句

status deleteclause(Total SATList, int temp); //删除符合要求的子句

status deleteword(Total SATList, int temp); //删除符合要求的文字

status addclause(Total SATList, int temp); //添加单子句

int judgeemptyclause(Total SATList); //判断子句集中是否含有空子句

int judgeemptytotal(Total SATList); //判断子句集是否为空

int getword1(Total SATList); //获取变元的2种算法

int getword2(Total SATList);

int getword3(Total SATList);

Total destroy(Total SATList); //销毁子句集

status DPLL1(Total SATList); //使用了获取变元算法一的DPLL算法

status DPLL2(Total SATList); //使用了获取变元算法二的DPLL算法

status saveanswer(Total SATList,int dresult, double time); //保存答案为res文件

status savecnf(Total SATList); //保存为cnf文件

//数独相关函数

void DPLLPRO(Total SATList); //返回解的个数的DPLL算法

status printsudoku(int sudoku[][10]); //打印数独

status convertresult(int sudoku[][10]); //结果数组转换为数独

status judgenum(int sudoku[][10], int row, int col, int n); //判断数是否可填

Total initial(Total SATList); //子句集初始化

status createnode(Total SATList,int temp); //创建文字

status createorigin(Total SATList, int sudoku[][10]); //原数独中不为0的数的创建

status createsingle(Total SATList, int sudoku[][10]); //单元格约束

status createrow(Total SATList, int sudoku[][10]); //行约束

status createcol(Total SATList, int sudoku[][10]); //列约束

status createblock(Total SATList, int sudoku[][10]); //分块约束

Total createcnf(Total SATList, int sudoku[][10]); //数独转化为SAT算例

status createsudoku(Total SATList, int sudoku[][10]); //创建数独

status createsudoku2(int sudoku[][10],int row,int num);

status digholes(int sudoku[][10], int count,int holes[][10]); //挖洞法创建待求解数独

#endif

**SAT.c的代码实现：**

#include "SAT\_SUDOKU.h"

Total fileload(Total SATList, char\* filename) //读取cnf文件

{

SATList = (Total)malloc(sizeof(total));

FILE\* fp;

char ch;

int temp;

if ((fp = fopen(filename, "r")) == NULL)

{

printf("文件打开失败！\n");

return NULL;

}

while (fread(&ch, sizeof(char), 1, fp)) //输出注释

{

if (ch != 'p') putchar(ch);

else

{

fread(&ch, sizeof(char), 1, fp);

if (ch == ' ') break;

}

}

for (int i = 0; i < 4; i++) //读入“ cnf ”

fread(&ch, sizeof(char), 1, fp);

fscanf(fp, "%d", &SATList->varcount); //读入文字数和公式数

fscanf(fp, "%d", &SATList->clacount);

printf("cnf文件中的文字数：%d\t公式数：%d\n", SATList->varcount, SATList->clacount);

varcount = SATList->varcount;

result = (int\*)malloc(sizeof(int) \* (SATList->varcount + 1));

for (int i = 1; i <= SATList->varcount; i++) result[i] = 0;

SATList->clahead = (Clause)malloc(sizeof(clause)); //以下到for循环皆为初始化

SATList->clahead->next = NULL;

SATList->clahead->head = (Varnode)malloc(sizeof(varnode));

SATList->clahead->head->next = NULL;

SATList->clahead->count\_of\_var = 0;

SATList->emptyclause = SATList->singleclause = 0;

SATList->count\_of\_clause = SATList->clacount;

Clause c = SATList->clahead;

for (int i = 1; fscanf(fp, "%d", &temp) != EOF && i <= SATList->clacount;)

{

if (temp != 0) //不为0说明这一句公式的读入没有完成

{

Varnode p = (Varnode)malloc(sizeof(varnode));

p->data = temp;

p->next = NULL;

Varnode q = c->head;

if (q->next == NULL && abs(q->data)> SATList->varcount) //首结点

c->head = p;

else { //其他结点

while (q->next) q = q->next;

q->next = p;

}

c->count\_of\_var++;

}

else //为0说明这句公式已经读完，i自增进行下一句公式的读入，这里为对下一句公式的初始化

{

if (i == SATList->clacount) //已读完，不必对下一句进行初始化

{

c->next = NULL;

break;

}

if (c->count\_of\_var == 1) SATList->singleclause++; //单子句个数加一

c->next = (Clause)malloc(sizeof(clause)); //以下为下一个语句的初始化

c->next->head = (Varnode)malloc(sizeof(varnode));

c->next->next = NULL;

c->next->head->next = NULL;

c->next->count\_of\_var = 0;

c = c->next;

i++;

}

}

printf("读取完毕!\n");

fclose(fp);

return SATList;

}

Total Duplicate(Total SATList) //复制子句集

{

Total temp = (Total)malloc(sizeof(total)); //副本的初始化

temp->varcount = SATList->varcount;

temp->clacount = SATList->clacount;

temp->singleclause = SATList->singleclause;

temp->emptyclause = SATList->emptyclause;

temp->count\_of\_clause = SATList->count\_of\_clause;

temp->clahead = (Clause)malloc(sizeof(clause));

temp->clahead->next = NULL;

temp->clahead->head = (Varnode)malloc(sizeof(varnode));

temp->clahead->head->next = NULL;

Clause c = SATList->clahead;

Clause ctemp = temp->clahead;

while (c) //遍历子句集，向副本中写入数据

{

ctemp->count\_of\_var = c->count\_of\_var;

Varnode p = c->head, ptemp = ctemp->head;

while (p)//遍历子句

{

ptemp->data = p->data;

p = p->next;

if (p == NULL) //最后一个文字

{

ptemp->next = NULL;

break;

}

ptemp->next = (Varnode)malloc(sizeof(varnode)); //下一个文字的初始化

ptemp = ptemp->next;

ptemp->next = NULL;

}

c = c->next;

if (c == NULL) //最后一句

{

ctemp->next = NULL;

break;

}

ctemp->next = (Clause)malloc(sizeof(clause)); //下一句的初始化

ctemp = ctemp->next;

ctemp->next = NULL;

ctemp->head = (Varnode)malloc(sizeof(varnode));

ctemp->head->next = NULL;

}

return temp;

}

status Traverse(Total SATList) //输出cnf文件

{

Clause c = SATList->clahead;

while (c) //遍历子句集

{

Varnode p = c->head;

printf("(");

while (p)

{

printf("%d", p->data);

if (p->next != NULL) printf("∨"); //不是最后一个文字的时候输出

p = p->next;

}

printf(")");

if (c->next != NULL) printf("∧"); //不是最后一句的时候输出

c = c->next;

}

return OK;

}

int findsingleclause(Total SATList) //寻找单子句

{

Clause c = SATList->clahead;

if (SATList->singleclause == 0) return 0; //没有单子句

while (c) //遍历子句集找第一个单子句

{

if (c->count\_of\_var == 1)

return c->head->data;

c = c->next;

}

return 0;

}

status deleteclause(Total SATList, int temp) //删除符合要求的子句

{

Clause c = SATList->clahead;

while (c) //遍历子句集

{

int flag = 0;

Varnode p = c->head,q;

while (p) //遍历子句

{

if (p->data == temp) //找到该文字

{

flag = 1; //改变的标记

if (c->count\_of\_var==1) SATList->singleclause--; //如果是单子句就让单子句数量减少

p = c->head;

q = p->next;

while (q) //销毁该子句除首结点之外的结点

{

p->next = p->next->next;

free(q);

q = p->next;

}

free(p); //销毁首结点

c->head = NULL;

SATList->count\_of\_clause--; //子句数量减少

if (c!=SATList->clahead) //c不是首子句，销毁c即可

{

Clause ctemp = SATList->clahead;

while (ctemp->next != c) ctemp = ctemp->next; //找到c的前一个子句

ctemp->next = c->next;

ctemp = ctemp->next;

free(c);

c = ctemp; //实际上c发生了后移

}

else if(c==SATList->clahead&&c->next==NULL) //c是首子句且只有c一个子句

{

free(SATList->clahead);

SATList->clahead = NULL;

}

else if (c == SATList->clahead && c->next != NULL) //c是首子句但还有其他子句

{

SATList->clahead = c->next; //首子句指向c的下一子句

free(c);

c = SATList->clahead; //c仍指向首子句

}

break;

}

p = p->next;

}

if (SATList->clahead == NULL) break; //子句集为空，直接返回

else

{

if (flag == 0) c = c->next; //未改变，c后移

}

}

return OK;

}

status deleteword(Total SATList, int temp) //删除符合要求的文字

{

Clause c = SATList->clahead;

while (c) //遍历子句集

{

int flag = 0;

Varnode p = c->head,q;

while (p) //遍历子句

{

if (p->data == -temp) //找到该文字

{

flag = 1; //改变的标记

c->count\_of\_var--; //该子句文字数减少

if (c->count\_of\_var == 0)

SATList->emptyclause++; //文字数为0，空子句数增加

if (p != c->head) //p不是首结点，销毁即可

{

q = c->head;

while (q->next != p) q = q->next;

q->next = p->next;

free(p);

q = q->next;

p = q; //实际上p发生后移

}

else if (p == c->head&&p->next!=NULL) //p是首结点但不是唯一一个结点

{

c->head = p->next;//首结点指向下一个结点

free(p);

p = c->head;

}

else if (p == c->head && p->next == NULL) //p是首结点且是唯一一个结点

{

free(c->head); //销毁首结点

c->head = NULL;

break;

}

continue; //继续遍历子句

}

p = p->next;

}

if (c->count\_of\_var == 1&&flag==1) SATList->singleclause++; //产生单子句，单子句数量增加

c = c->next;

}

return OK;

}

status addclause(Total SATList, int temp) //增加单子句

{

Clause c = SATList->clahead;

while (c->next) c = c->next;//找到最后一个子句

Clause ctemp = (Clause)malloc(sizeof(clause));

ctemp->count\_of\_var = 1;

ctemp->next = NULL;

ctemp->head = (Varnode)malloc(sizeof(varnode));

ctemp->head->data = temp;

ctemp->head->next = NULL;

c->next = ctemp;

SATList->count\_of\_clause++;

SATList->singleclause++;

return OK;

}

int judgeemptyclause(Total SATList) //判断子句集中是否含有空子句

{

if (SATList->emptyclause) return 1;

return 0;

}

int judgeemptytotal(Total SATList) //判断子句集是否为空

{

if (SATList->count\_of\_clause == 0) return 1;

return 0;

}

int getword1(Total SATList) //获取变元算法一

{

int pmax = 0,nmax = 0,ptemp = 0,ntemp = 0,temp=0;

int \*pfre=(int \*)malloc(sizeof(int)\*(SATList->varcount+1));

int \* nfre = (int\*)malloc(sizeof(int) \* (SATList->varcount + 1));

for (int i = 1; i <= SATList->varcount; i++)

pfre[i] =nfre[i]= 0;

Clause c = SATList->clahead;

while (c) //统计所有正负变元的出现次数

{

Varnode p = c->head;

while (p)

{

if (p->data > 0) pfre[p->data]++;

else nfre[-p->data]++;

p = p->next;

}

c = c->next;

}

for (int i = 1; i <= SATList->varcount; i++)

{

if (pmax < pfre[i])

{

pmax = pfre[i];

ptemp = i;

}

if (nmax < nfre[i])

{

nmax = nfre[i];

ntemp = i;

}

}

free(pfre);

free(nfre);

return temp=pmax>nmax?ptemp:ntemp; //返回出现次数最多的文字

}

int getword2(Total SATList) //获取变元算法二

{

return SATList->clahead->head->data; //找第一个子句的第一个文字

}

int getword3(Total SATList)

{

double pmax = 0, nmax = 0 ;

int ptemp = 0, ntemp = 0, temp = 0;

double\* pfre = (double\*)malloc(sizeof(double) \* (SATList->varcount + 1));

double\* nfre = (double\*)malloc(sizeof(double) \* (SATList->varcount + 1));

for (int i = 1; i <= SATList->varcount; i++)

pfre[i] = nfre[i] = 0;

Clause c = SATList->clahead;

while (c) //统计所有正负变元的出现次数

{

Varnode p = c->head;

while (p)

{

if (p->data > 0) pfre[p->data]+=pow(2,-c->count\_of\_var);

else nfre[-p->data]+= pow(2, -c->count\_of\_var);

p = p->next;

}

c = c->next;

}

for (int i = 1; i <= SATList->varcount; i++)

{

if (pmax < pfre[i])

{

pmax = pfre[i];

ptemp = i;

}

if (nmax < nfre[i])

{

nmax = nfre[i];

ntemp = -i;

}

}

free(pfre);

free(nfre);

return temp = pmax > nmax ? ptemp : ntemp; //返回出现次数最多的文字

}

Total destroy(Total SATList)

{

Clause c = SATList->clahead;

while (c) //遍历子句集

{

if (c->head != NULL) //销毁子句的所有文字

{

Varnode p = c->head, q = p->next;

while (q)

{

p->next = p->next->next;

free(q);

q = p->next;

}

free(p);

c->head = NULL;

}

c = c->next;

}

c = SATList->clahead;

if (c != NULL) //销毁所有子句结点

{

Clause ctemp = c->next;

while (ctemp)

{

c->next = c->next->next;

free(ctemp);

ctemp = c->next;

}

free(c);

SATList->clahead = NULL;

}

free(SATList); //销毁子句集

SATList = NULL;

return SATList;

}

status DPLL1(Total SATList) //核心算法

{

int temp;

temp = findsingleclause(SATList); //找单子句

while (temp)

{

if (temp > 0) result[temp] = 1; //更新结果数组

else result[-temp] = 0;

deleteclause(SATList, temp); //删除子句和文字

deleteword(SATList, temp);

if (judgeemptytotal(SATList)) //判断是否为空

{

SATList=destroy(SATList);

return OK;

}

if (judgeemptyclause(SATList)) //判断有无空子句

{

SATList=destroy(SATList);

return ERROR;

}

temp = findsingleclause(SATList); //找单子句

}

temp = getword1(SATList); //获取变元算法一

Total ctemp = Duplicate(SATList); //复制子句集，用于回溯

addclause(SATList, temp); //添加变元子句

if (DPLL1(SATList)) //递归调用

return OK;

else {

addclause(ctemp, -temp); //添加负变元子句

return DPLL1(ctemp); //递归调用

}

}

status DPLL2(Total SATList)

{

int temp;

temp = findsingleclause(SATList);

while (temp)

{

if (temp > 0) result[temp] = 1;

else result[-temp] = 0;

deleteclause(SATList, temp);

deleteword(SATList, temp);

if (judgeemptytotal(SATList))

{

SATList=destroy(SATList);

return OK;

}

if (judgeemptyclause(SATList))

{

SATList=destroy(SATList);

return ERROR;

}

temp = findsingleclause(SATList);

}

temp = getword2(SATList); //获取变元算法二

Total ctemp = Duplicate(SATList);

addclause(SATList, temp);

if (DPLL2(SATList))

return OK;

else {

addclause(ctemp, -temp);

return DPLL2(ctemp);

}

}

void DPLLPRO(Total SATList)

{

if (allresult > 1) return;

int temp;

temp = findsingleclause(SATList);

while (temp)

{

if (temp > 0) result[temp] = 1;

else result[-temp] = 0;

deleteclause(SATList, temp);

deleteword(SATList, temp);

if (judgeemptytotal(SATList))

{

SATList=destroy(SATList);

allresult++;

return;

}

if (judgeemptyclause(SATList))

{

SATList=destroy(SATList);

return;

}

temp = findsingleclause(SATList);

}

temp = getword2(SATList);

Total ctemp = Duplicate(SATList);

addclause(SATList, temp);

DPLLPRO(SATList); //两种情况分别考虑

addclause(ctemp, -temp);

DPLLPRO(ctemp);

}

status saveanswer(Total SATList,int dresult, double time) //保存为同名res文件

{

int len = strlen(savefile);

savefile[len - 1] = 's'; //更改后缀

savefile[len - 2] = 'e';

savefile[len - 3] = 'r';

FILE\* fp;

if ((fp = fopen(savefile, "wb")) == NULL)

{

printf("文件保存失败\n");

return ERROR;

}

else

{

if (dresult == 0 || dresult == 1)

{

fprintf(fp, "s %d \n", dresult);

if (dresult == 1)

{

fprintf(fp, "v ");

for (int i = 1; i <= varcount; i++)

fprintf(fp, "%d ", (result[i] == 1) ? i : -i);

fprintf(fp, "\nt %.0lf", time);

}

}

else

fprintf(fp, "-1");

printf("文件保存成功\n");

return OK;

}

}

status savecnf(Total SATList) //保存为cnf文件

{

char filename[100];

printf("请输入保存的文件名：\n");

scanf("%s", filename);

FILE\* fp;

if ((fp = fopen(filename, "wb")) == NULL)

{

printf("文件保存失败\n");

return ERROR;

}

else

{

fprintf(fp, "p cnf ");

fprintf(fp,"%d %d\n", SATList->varcount, SATList->clacount);

Clause c = SATList->clahead;

while (c)

{

Varnode p = c->head;

while (p)

{

fprintf(fp, "%d ", p->data);

p = p->next;

}

fprintf(fp, "0\n");

c = c->next;

}

printf("文件保存成功\n");

return OK;

}

}

**SUDOKU.c的代码实现：**

#include "SAT\_SUDOKU.h"

status printsudoku(int sudoku[][10]) //输出数独

{

for (int i = 1; i <= 9; i++)

{

if(i%3==1) printf("-------------------------------\n"); //分割线

for (int j = 1; j <= 9; j++)

{

if (j % 3 == 1) printf("|"); //分割线

printf("%2d ", sudoku[i][j]);

if (j == 9) printf("|\n"); //分割线

}

}

printf("-------------------------------\n"); //分割线

return OK;

}

status convertresult( int sudoku[][10]) //结果转换

{

for (int t = 1; t <= 729; t++)

if (result[t] == 1)

for (int i = 1; i <= 9; i++)

for (int j = 1; j <= 9; j++)

for (int n = 1; n <= 9; n++)

if (t == (i - 1) \* 81 + (j - 1) \* 9 + n)

sudoku[i][j] = n;

return OK;

}

status judgenum(int sudoku[][10], int row, int col, int n) //判断填入该数是否可行

{

int temp1, temp2;

for (int k = 1; k <= 9; k++)

if (sudoku[row][k] == n || sudoku[k][col] == n) return ERROR; //所在行和列不能有重复

for (int k = 1; k <= 9; k++) //找到所在的分块

if (k >= row && k % 3 == 0)

{

temp1 = k;

break;

}

for(int k=1;k<=9;k++)

if (k >= col && k % 3 == 0)

{

temp2 = k;

break;

}

for (int i = temp1 - 2; i <= temp1; i++) //分块中不能有重复

for (int j = temp2 - 2; j <= temp2; j++)

if (sudoku[i][j] == n) return ERROR;

return OK;

}

Total initial(Total SATList) //初始化子句集

{

SATList = (Total)malloc(sizeof(total));

result = (int\*)malloc(sizeof(int) \* 730);

for (int i = 1; i <= 729; i++) result[i] = 0;

SATList->count\_of\_clause=SATList->emptyclause=SATList->singleclause= SATList->clacount = 0;

SATList->varcount = 729;

SATList->clahead = (Clause)malloc(sizeof(clause));

SATList->clahead->next = NULL;

SATList->clahead->count\_of\_var = 0;

SATList->clahead->head = (Varnode)malloc(sizeof(varnode));

SATList->clahead->head->next = NULL;

return SATList;

}

status createnode(Total SATList,int temp) //创建文字

{

Clause c = SATList->clahead;

while (c->next) c = c->next;

if (temp != 0) //不为0说明这一句公式的读入没有完成

{

Varnode p = (Varnode)malloc(sizeof(varnode));

p->data = temp;

p->next = NULL;

Varnode q = c->head;

if (q->next == NULL && abs(q->data) > SATList->varcount) c->head = p; //首结点

else { //非首结点

while (q->next) q = q->next;

q->next = p;

}

c->count\_of\_var++;

}

else //为0说明这句公式已经读完，这里为对下一句公式的初始化

{

SATList->clacount++;

SATList->count\_of\_clause++;

if (c->count\_of\_var == 1) SATList->singleclause++;

c->next = (Clause)malloc(sizeof(clause));

c->next->head = (Varnode)malloc(sizeof(varnode));

c->next->next = NULL;

c->next->head->next = NULL;

c->next->count\_of\_var = 0;

}

return OK;

}

status createorigin(Total SATList, int sudoku[][10]) //创建原数独中不为0的数对应的文字

{

for(int i=1;i<=9;i++)

for(int j=1;j<=9;j++)

if (sudoku[i][j])

{

createnode(SATList, (i - 1) \* 81 + (j - 1) \* 9 + sudoku[i][j]); //转换公式

createnode(SATList, 0); //结束标志

}

return OK;

}

status createsingle(Total SATList, int sudoku[][10]) //单元格约束

{

for(int i=1;i<=9;i++)

for(int j=1;j<=9;j++)

if (sudoku[i][j] == 0) //该单元格可以至少填入一个数字

{

for (int k = 1; k <= 9; k++)

createnode(SATList, (i-1)\*81+(j-1)\*9+k);

createnode(SATList, 0);

for(int m=1;m<=9;m++)

for(int n=m+1;n<=9;n++) //该单元格只能填入一个数字

{

createnode(SATList, -((i - 1) \* 81 + (j - 1) \* 9 + m));

createnode(SATList, -((i - 1) \* 81 + (j - 1) \* 9 + n));

createnode(SATList, 0);

}

}

return OK;

}

status createrow(Total SATList, int sudoku[][10]) //行约束

{

for (int row = 1; row <= 9; row++)

{

for (int j = 1; j <= 9; j++) //该行含有1-9

{

for (int i = 1; i <= 9; i++)

createnode(SATList, (row - 1) \* 81 + (i - 1) \* 9 + j);

createnode(SATList, 0);

}

for (int n = 1; n <= 9; n++) //该行的每个位置只能填入一个数字

for(int i=1;i<=9;i++)

for (int j = i + 1; j <= 9; j++)

{

createnode(SATList, -((row - 1) \* 81 + (i - 1) \* 9 + n));

createnode(SATList, -((row - 1) \* 81 + (j - 1) \* 9 + n));

createnode(SATList, 0);

}

}

return OK;

}

status createcol(Total SATList, int sudoku[][10]) //列约束

{

for (int col = 1; col <= 9; col++)

{

for (int i = 1; i <= 9; i++)//该列含有1-9

{

for (int j = 1; j <= 9; j++)

createnode(SATList, (j - 1) \* 81 + (col - 1) \* 9 + i);

createnode(SATList, 0);

}

for (int n = 1; n <= 9; n++) //该列的每个位置只能填入一个数字

for (int i = 1; i <= 9; i++)

for (int j = i + 1; j <= 9; j++)

{

createnode(SATList, -((i - 1) \* 81 + (col - 1) \* 9 + n));

createnode(SATList, -((j - 1) \* 81 + (col - 1) \* 9 + n));

createnode(SATList, 0);

}

}

return OK;

}

status createblock(Total SATList, int sudoku[][10]) //分块约束

{

for (int x = 1; x <= 3; x++)

{

for (int y = 1; y <= 3; y++)

{

int erow = x \* 3, ecol = y \* 3; //找到所在分块

int brow = erow - 2, bcol = ecol - 2;

for (int n = 1; n <= 9; n++) //该分块可以填入1-9

{

for (int i = brow; i <= erow; i++)

for (int j = bcol; j <= ecol; j++)

createnode(SATList, (i - 1) \* 81 + (j - 1) \* 9 + n);

createnode(SATList, 0);

}

for (int k = 1; k <= 9; k++) //对分块中每个单元格，只能填入一个数字

{

for(int x= brow;x<=erow;x++)

for (int y = bcol; y <= ecol; y++)

{

for(int m=x;m<=erow;m++)

for (int n = (m==x?y+1:bcol); n <= ecol; n++)

{

createnode(SATList, -((x - 1) \* 81 + (y - 1) \* 9 + k));

createnode(SATList, -((m - 1) \* 81 + (n - 1) \* 9 + k));

createnode(SATList, 0);

}

}

}

}

}

return OK;

}

Total createcnf(Total SATList, int sudoku[][10]) //将数独转换为SAT算例

{

SATList=initial(SATList);

createorigin(SATList, sudoku);

createsingle(SATList, sudoku);

createrow(SATList, sudoku);

createcol(SATList, sudoku);

createblock(SATList, sudoku);

Clause c = SATList->clahead;

while (c->next->next) c = c->next; //最后一个子句的next指向空，因为多初始化了一次

free(c->next);

c->next = NULL;

return SATList;

}

status createsudoku(Total SATList, int sudoku[][10]) //创建完整答案数独

{

srand(time(0));

int count = 1;

while (count <= 11) //随机赋值

{

int i, j, n;

i = rand() % 9 + 1;

j = rand() % 9 + 1;

n = rand() % 9 + 1;

if (sudoku[i][j] == 0 && judgenum(sudoku, i, j, n))

{

sudoku[i][j] = n;

count++;

}

}

SATList=createcnf(SATList, sudoku); //转化为SAT算例

DPLL1(SATList); //求解

convertresult(SATList, sudoku); //转化为数独

return OK;

}

status createsudoku2(int sudoku[][10],int row,int num)

{

int c[10];

srand(time(0));

for (int i = 1; i <= 9; i++)

c[i] = i;

for (int i = 1; i <= 20; i++)

{

int k = rand() % 9 + 1;

int temp = c[1];

c[1] = c[k];

c[k] = temp;

}

for (int i = 1; i <= 9; i++)

{

int col = c[i];

if (judgenum(sudoku, row, col, num) && !sudoku[row][col])

{

sudoku[row][col] = num;

if (row == 9)

{

if (num == 9) return OK;

if (createsudoku2(sudoku, 1, num + 1)) return OK;

}

else

{

if (createsudoku2(sudoku, row + 1, num)) return OK;

}

sudoku[row][col] = 0;

}

}

return ERROR;

}

status digholes(int sudoku[][10], int count,int holes[][10]) //挖洞法创建需求解的数独

{

srand(time(0));

do

{

for(int i=1;i<=9;i++)

for(int j=1;j<=9;j++)

if (holes[i][j])

{

sudoku[i][j] = holes[i][j];

holes[i][j] = 0;

}

int cnt = count;

while (cnt)

{

int x = rand() % 9 + 1;

int y = rand() % 9 + 1;

if (sudoku[x][y] != 0)

{

holes[x][y] = sudoku[x][y];

sudoku[x][y] = 0;

cnt--;

}

}

Total SATList = (Total)malloc(sizeof(total));

SATList = createcnf(SATList, sudoku);

allresult = 0;

DPLLPRO(SATList);

} while (allresult > 1);

printf("\n需要求解的数独为：\n");

printsudoku(sudoku);

}

**main.c的代码实现：**

#include "SAT\_SUDOKU.h"

int main() {

srand(time(0));

Total SATList = NULL;

int sudoku[10][10] = { 0 }, dsudoku[10][10] = { 0 }, holes[10][10] = { 0 }, origin[100] = { 0 };

FILE \*fp;

int op = 1, op1 = 1, op2 = 1, op3 = 1, dresult, temp1, temp2, temp3, holescount;

double t1, t2;

int n, m, k, i, j, flag = 0,flag1=0;

while (op) { //一级菜单

system("cls");

printf("\n Menu for SUDOKU or SAT \n");

printf("------------------------------------------------------\n");

printf(" 1.SAT 2.SUDOKU \n");

printf(" 0.退出 \n");

printf("------------------------------------------------------\n");

printf("请输入操作（0-2）：\n");

scanf("%d", &op);

switch (op) {

case 1: {

op1 = 1;

while (op1) { //二级菜单1

system("cls");

printf("\n Menu for SAT \n");

printf("--------------------------------------------\n");

printf(" 1.读入cnf文件 2.输出当前子句集 \n");

printf(" 3.求解并保存答案 4.保存为cnf文件 \n");

printf(" 5.销毁当前子句集 0.退出\n");

printf("--------------------------------------------\n");

scanf("%d", &op1);

switch (op1) {

case 1: {

if (SATList != NULL)

printf("已读入cnf文件，不能重复读入\n");

else {

printf("输入文件名: ");

scanf("%s", filename);

strcpy(savefile, filename);//保存副本

SATList = fileload(SATList, filename);

}

system("pause");

break;

}

case 2: {

if (SATList == NULL)

printf("未读入cnf文件\n");

else {

temp1 = Traverse(SATList);

if (temp1 == 1)

printf("输出成功\n");

else

printf("输出失败\n");

}

system("pause");

break;

}

case 3: {

if (SATList == NULL)

printf("未读入cnf文件\n");

else {

dresult = -1;

printf("求解完成后自动销毁生成的子句集\n请选择变元选取策略：\n1.第一子句的第一文字 2.优化算法\n");

scanf("%d", &temp3);

if (temp3 == 1) {

t1 = clock();

dresult = DPLL2(SATList);

t2 = clock();

SATList = NULL;

}

else if (temp3 == 2) {

t1 = clock();

dresult = DPLL1(SATList);

t2 = clock();

SATList = NULL;

}

printf("求解完成\n\n");

if (dresult) {

printf("其中一个答案如下：\n");

for (int i = 1; i <= varcount; i++) {

if (result[i] == 1)

printf("%d ", i);

else

printf("%d ", -i);

}

printf("\n\n求解所用时间为%lfms\n\n", t2 - t1);

} else if(dresult==0){

printf("该算例无解\n");

printf("\n求解所用时间为%lfms\n\n", t2 - t1);

}

saveanswer(SATList, dresult, t2 - t1);

}

system("pause");

break;

}

case 4: {

if (SATList == NULL)

printf("未读入cnf文件\n");

else

savecnf(SATList);

system("pause");

break;

}

case 5: {

if (SATList == NULL)

printf("子句集不存在\n");

else

{

SATList = destroy(SATList);

printf("销毁完成\n");

}

system("pause");

break;

}

case 0:

break;

default:

printf("输入错误\n");

system("pause");

break;

}

}

break;

}

case 2: {

op2 = 1;

while (op2) {

//二级菜单2

system("cls");

printf("\n Menu for Sudoku \n");

printf("-------------------------------------------\n");

printf(" 1.生成数独 2.开始游戏 \n");

printf(" 0.退出 \n");

printf("-------------------------------------------\n");

printf("请输入操作（0-2）：\n");

scanf("%d", &op2);

switch (op2) {

case 1: {

memset(sudoku, 0, sizeof(sudoku));

memset(holes, 0, sizeof(holes));

memset(dsudoku, 0, sizeof(dsudoku));

memset(origin, 0, sizeof(origin));

createsudoku2(sudoku, 1, 1);

printf("\n数独终盘为：\n");

printsudoku(sudoku);

printf("请输入挖洞数量（数量越多，难度越高，生成待求解数独越慢）:\n"); //在已生成的数独的基础上挖洞

scanf("%d", &holescount);

for (i = 1; i <= 9; i++)

for (j = 1; j <= 9; j++)

dsudoku[i][j] = sudoku[i][j];

allresult = 0;

digholes(dsudoku, holescount, holes);

system("pause");

break;

}

case 2: {

flag1 = 0;

for (i = 1; i <= 9; i++)

{

for (j = 1; j <= 9; j++)

if (dsudoku[i][j] == 0)

{

flag1 = 1;

break;

}

if (flag1 == 1) break;

}

if (flag1 == 0)

{

printf("未生成数独！\n");

system("pause");

}

else

{

op3 = 1;

while (op3) {

flag = 0;

system("cls");

printsudoku(dsudoku);

printf("可操作位置为：\n");

int count = 0;

for (i = 1; i <= 9; i++)

for (j = 1; j <= 9; j++)

if (holes[i][j]) {

printf("%d %d ", i, j);

origin[i \* 10 + j] = 1;

count++;

if (count == 7) {

printf("\n");

count = 0;

}

}

printf("\n");

printf("1.填写数字 2.删除数字 \n"); //开始游戏

printf("3.查看答案并退出 0.退出 \n");

printf("请输入操作（0-3）：\n");

scanf("%d", &op3);

if (op3 == 1) {

printf("请输入你想填入的位置（依次输入行序号1-9和列序号1-9）：\n");

scanf("%d%d", &n, &m);

if (m >= 1 && m <= 9 && n >= 1 && n <= 9 && dsudoku[n][m] != 0) {

printf("该位置不为空,请重新操作\n");

system("pause");

}

else if (m >= 1 && m <= 9 && n >= 1 && n <= 9 && dsudoku[n][m] == 0) {

printf("请选择你想填入的数值（1-9）：\n");

scanf("%d", &k);

dsudoku[n][m] = k;

}

else if (m < 1 || m > 9 || n < 1 || n > 9) {

printf("输入位置错误\n");

system("pause");

}

}

else if (op3 == 2) {

printf("请输入你想删除的位置（依次输入行序号1-9和列序号1-9）:\n");

scanf("%d%d", &n, &m);

if (m >= 1 && m <= 9 && n >= 1 && n <= 9) {

if (dsudoku[n][m] == 0) {

printf("该位置已经为空，不能删除\n");

system("pause");

}

else if (origin[n \* 10 + m] == 0)

{

printf("该位置为原始数据，不能删除\n");

system("pause");

}

else dsudoku[n][m] = 0;

}

else {

printf("输入位置有误");

system("pause");

}

}

else if (op3 == 3) {

for (i = 1; i <= 9; i++) {

//判断用户的填入是否正确

for (j = 1; j <= 9; j++)

if (dsudoku[i][j] != sudoku[i][j]) {

flag = 1;

break;

}

if (flag == 1)

break;

}

if (flag == 1) {

printf("你填写的数独错误！");

printf("数独正确的解为：\n");

}

else

printf("你填入的解正确！\n");

printsudoku(sudoku);//输出正确的数独解

system("pause");

break;

}

else if (op3 == 0)

break;

else {

printf("输入错误\n");

system("pause");

}

}

}

break;

}

case 0:

break;

default:

printf("输入错误\n");

system("pause");

break;

}

}

break;

}

case 0:

break;

default:

printf("输入错误\n");

system("pause");

break;

}//end of switch

}//end of while

printf("欢迎下次使用本系统！\n");

system("pause");

return 0;

}//end of main()